



A GUIDE FOR STUDENTS & BOOTCAMPERS

What happens to the junior developer?

A guide for students, bootcampers, and anyone considering the profession in the agent era.

AUTHOR

Jason Cruz Falzon

Creator of the Cognitive Delivery Framework

What happens to the junior developer?

The question I get asked most often, usually by students approaching graduation or by people considering a switch into tech, is some version of this: if AI agents can write the code, is it still worth becoming a software developer?

The honest answer. Yes. More than that, this might be one of the most interesting moments in the history of the profession to enter it. The conventional wisdom says otherwise. The conventional wisdom is wrong.

WHY NOW IS GENUINELY A GOOD TIME TO START

The first thing worth knowing is that the field is still growing. The US Bureau of Labor Statistics projects software developer employment to grow 15 per cent between 2024 and 2034, against an all-occupation average of three per cent, with around 129,200 openings each year over the decade.¹ The BLS explicitly cites continued expansion of AI, IoT, robotics and security as drivers of that demand, not threats to it.

The shape of the work has changed before. It changed when compilers replaced assembler. It changed when frameworks replaced bare metal. It changed when the cloud replaced the server room. Every shift compressed something at the bottom of the stack and opened something at the top. This shift is bigger than those, but it follows the same pattern.

What it means for someone starting now is this. The advantages of long-tenured engineers, fluent typing and encyclopaedic memory of API quirks, matter less than they used to. The advantages of judgement, communication, system thinking and curiosity matter more. Those are not advantages locked behind a decade of experience. They are advantages anyone can start cultivating today.

There is also a quieter thing happening. Smaller teams can now build things that used to require headcount. The cost of trying something has fallen. More startups, more side projects, more shots on goal. The people who benefit most from that are the ones starting their careers now, with no commitment to the old way of working.

The ceiling has not fallen. The floor has risen. The trajectory between them has changed.

WHAT IS ACTUALLY CHANGING

For most of the history of software, junior developers earned their stripes by writing a lot of code. Boilerplate, glue, tests, simple features. The work was repetitive on purpose. It was how you built fluency. Seniors set direction, juniors produced the volume, and somewhere in the years between joining and being trusted, judgement accumulated.

Agents now produce that volume in seconds. The base of the pyramid has not vanished, but it has been compressed. What used to be a year of learning by doing can be a few weeks of learning by reviewing. That forces a different question: what is a junior developer actually for?

The answer in a hybrid human-agent delivery model is something like this. Less typing, more reading. Less producing code, more deciding whether the code in front of you is fit for purpose. Less following the pattern, more asking whether the pattern the agent reached for fits the context. Less writing, more verifying. Less coding, more thinking.

This is genuinely harder than what came before. It is also more interesting.

THE SHAPE OF THE ROLE	THEN (PRE-AGENT)	NOW (WITH AGENTS)
The work	Writing code	Reviewing and verifying
The bottleneck	Code production	Code verification
A junior's day	Boilerplate, glue, simple features	Reading diffs, checking output
What gets you hired	Years of practice	Evidence of judgement
Career inflection	Roughly ten years in	The first two years

BY THE NUMBERS

+15%

projected growth in software developer jobs, 2024 to 2034

US BLS, 2025

84%

of developers use or plan to use AI tools, up from 76% in 2024

Stack Overflow, 2025

29%

of developers trust AI accuracy, down from 40% the year before

Stack Overflow, 2025

45%

of AI-generated code introduces an OWASP Top 10 vulnerability

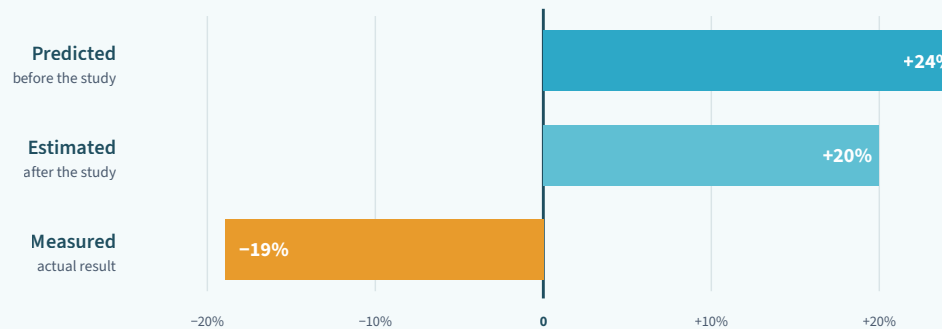
Veracode, 2025

THE VERIFICATION GAP, AND WHY IT MATTERS FOR YOU

The single most important finding in the research from the past year, for someone entering the profession, is this. In a randomised controlled trial published by METR in mid-2025, sixteen experienced open-source developers worked on real tasks in their own projects. Some tasks were assigned with AI tools, some without. Before the study, they predicted AI would speed them up by 24 per cent. After the study, they estimated it had sped them up by 20 per cent. In reality, AI had slowed them down by 19 per cent.²

THE PERCEPTION GAP

What developers **predicted**, what they **believed** afterwards, and what was actually **measured**.



Headline figures from METR's mid-2025 randomised controlled trial. Their February 2026 follow-up suggests the magnitude of the slowdown has narrowed for later AI tools, but the perception gap itself remains the durable finding.

That gap, between perceived speed and measured speed, is the most important number in this guide. It says something quite specific about how AI tools work in practice. They feel productive even when they are not. They produce confident output that masks the cost of verifying it.

This shows up in the wider data. Stack Overflow's 2025 Developer Survey, which had 49,000 responses across 177 countries, found that while 84 per cent of developers use or plan to use AI tools, only 29 per cent trust their accuracy, down from 40 per cent the year before.³ Sixty-six per cent say they are losing time on AI solutions that are almost right but not quite. Forty-five per cent say debugging AI-generated code is more time-consuming than writing it themselves.

Veracode tested over 100 large language models on 80 coding tasks. Forty-five per cent of the AI-generated code introduced vulnerabilities aligned with the OWASP Top 10, the most critical web application security risks.⁴ The numbers were worse for some languages. Java reached a 72 per cent security failure rate.

None of this means AI tools are not useful. It means the value goes to people who can verify what comes out of them. That is the single best argument for learning the foundations properly, and for entering the profession now.

THE FOUNDATIONS TO LEARN FIRST

Before anything else, the foundations. These do not change. They are what made software engineers valuable before the agent era, and they are what makes you able to verify, govern and direct an agent now. Skip these and you will spend your career renting confidence from a tool you do not understand.

A programming language you can think in. Python is the most accessible starting point. Stack Overflow's 2025 survey shows Python's adoption rose seven percentage points in a single year, the largest jump for any major language, driven by AI, data science and back-end use.³ JavaScript or TypeScript if your interest is the

web. Pick one. Learn it well enough to read someone else's code without copying it into a chat window. Breadth across many languages is less valuable than depth in one.

Git. Every team uses it. Most graduates use it badly. Learn what a commit is, why branches exist, what a merge conflict actually means, how to recover from your own mistakes. Read the manual. The graduates who arrive Git-fluent stand out immediately.

The terminal. You will spend more time here than you expect. Learn to navigate, find files, search through text, kill processes, edit configuration. Comfort with the command line is one of the clearest signals that someone has actually written software, not just talked about it.

SQL. Almost every system stores data in a relational database. SELECT, JOIN, GROUP BY. The basics carry you a long way, and they age beautifully.

HTTP, APIs, and the request-response cycle. Almost everything modern is built on this. Understand how a request leaves the browser, hits a server, and returns data. Know what 200, 404 and 500 mean.

One framework, properly. React for the front end, FastAPI or Express for the back end, Django or Rails if you want full stack. Pick one and learn it deeply rather than skim three. Recruiters tell you they want breadth. Engineers know depth wins.

Testing. What a unit test is, what an integration test is, why they exist. Write them. Treat tests as part of the deliverable, not as paperwork. The 45 per cent vulnerability finding from Veracode means tests matter more, not less, when AI is in the mix.⁴

Reading. This is the meta-skill. Read documentation. Read other people's code. Read pull requests on projects you respect. The single biggest predictor of how fast someone gets good is how much code they read. Most graduates underweight this. Do not.

BEYOND THE FOUNDATIONS

Once the foundations are in place, the layer that distinguishes you in the agent era looks different to the layer that used to distinguish you. It is less about technical breadth, more about judgement. The Stack Overflow survey found that, even in a hypothetical future where AI can do most coding tasks, 75 per cent of developers say they would still ask another person for help when they do not trust AI's answers.³ That is the work. That is the job.

THE LAYER THAT MATTERS NOW

Code review craft. The ability to read a diff and see what is missing, risky or assumed. This used to be a senior skill. It is now a graduate skill.

Working with agents. Brief precisely, verify before trusting, push back when output does not fit context. Treat the agent as a colleague to manage, not a tool to use.

Governance literacy. Knowing what an agent should and should not decide, where accountability attaches, and how to keep the audit trail clean.

Domain thinking. Learn the business your software runs, not just the syntax it is written in. Pick an industry you find interesting and go deep.

Communication. Explain a trade-off in one sentence. Disagree clearly without being defensive. Write pull requests that age well. Ask the question instead of bluffing.

These are harder to develop than the foundations. They take years. They are also what makes a career, not just a first job.

AN HONEST WORD ABOUT THE JOB MARKET

The path in is harder than it was five years ago. HackerRank's 2025 Developer Skills Report, which surveyed 13,732 developers, engineering managers and recruiters across 102 countries, found that 74 per cent of developers say finding a job is difficult, with junior hiring lagging behind more experienced roles.⁵ Some of that is hesitation from employers about whether early-career developers can code without heavy AI assistance. Some of it is companies wanting hires who can deliver immediate value rather than be trained.

This is not a reason to back out. It is a reason to prepare differently. Companies are not less interested in talent. They are more interested in evidence. The graduates who get hired in this market are the ones who walk in with proof of work, not just credentials. The next section is what that preparation looks like.

WHAT TO DO BEFORE YOU START

If you have not yet got a job in the field, here is what to do today.

MOVE ON THESE NOW

- 1 **Build something and ship it.** A small website, a CLI tool, a Discord bot, a data scraper. Anything. Make it work end to end. Deploy it. Put the link on your CV. Finishing matters more than the technology choice.
- 2 **Contribute to open source, even tiny things.** Fix a typo in a README. Improve documentation. Add a test case. Most maintainers will guide you through your first PR. The contribution lives on your GitHub forever.
- 3 **Read pull requests in real projects.** Pick something you find interesting. Read its recent PRs. Work out why one was rejected and another merged. The cheapest way in the world to learn what good code review looks like.
- 4 **Use agents deliberately, not as a shortcut.** Brief them on real work. Verify their output. Push back when it is wrong. Every interaction is a rehearsal for how you will work in your first job.
- 5 **Learn one domain alongside the technical skill.** Finance, healthcare, logistics, education, climate. Whichever you find genuinely interesting. The graduates who pull ahead in the next five years will be the ones who understand the business their software runs.

THE TRAPS TO AVOID

Patterns I see go wrong over and over. Avoid these and you are ahead of most people in your cohort.

Using agents to skip learning the foundations. It feels productive. It is borrowed productivity. The METR data found that developers most familiar with their codebases were slowed down most by AI tools, partly because they could see when the AI was wrong and partly because correcting it took time the AI could not save.² If you skip the foundations, you cannot see when the AI is wrong. That is the worst position to be in.

Optimising the CV for keywords. Listing seventeen technologies you have touched once each. Recruiters now search for depth signals: a finished project, a real contribution to a real codebase, evidence you stayed somewhere long enough to learn it.

Imitating without understanding. Copying a pattern from Stack Overflow or an agent without knowing why it works. Every shortcut here becomes a hole in your foundation that shows up later, usually under pressure.

Bootcamp without portfolio. A bootcamp certificate by itself is no longer enough. The certificate plus three things you have built and shipped is.

Faking it through reviews. Approving a pull request you do not really understand. The most damaging habit you can develop. It corrodes trust, and trust is the currency of senior roles.

Treating the first job as the destination. It is the foundation. The graduates who plan three jobs ahead, not just one, end up where they want to be.

THE CLOSING THOUGHT

The question is not whether software development is still worth entering. It is whether you are willing to enter it on its new terms. Less typing, more thinking. Less producing code, more deciding whether the code in front of you is fit for purpose. The data shows a profession still expanding, with tools that are powerful but unreliable, and a market that is rewarding evidence of real skill more than ever. Most of the engineers I respect are excited about the new shape of the work, not nervous about it. There is a reason for that.

If you are willing to think clearly, communicate honestly, verify everything, and stay curious, this is a profession worth entering. The bar is high. The ceiling is higher.

WHAT FOLLOWS

The next page lists every source cited in this guide. The two pages after that are a Field Kit you can save, print or screenshot. **Prepare** is the foundations and projects to build before you apply. **Choose** is what to look for in the teams that interview you.

References

- 1 US Bureau of Labor Statistics.** Occupational Outlook Handbook: Software Developers, Quality Assurance Analysts, and Testers. Employment projections 2024 to 2034, published August 2025. Headline figures: 15 per cent projected employment growth, around 129,200 annual openings, 1.7 million jobs in 2024.
bls.gov/ooh/computer-and-information-technology/software-developers.htm
- 2 Becker, J., Rush, N., Barnes, B., and Rein, D. (METR).** Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. Randomised controlled trial, 16 developers, 246 tasks across mature open-source repositories. Headline finding: AI tools made experienced developers 19 per cent slower, while developers themselves believed they had been sped up by 20 per cent. METR has since acknowledged that later 2025 AI tools likely show smaller slowdowns or modest speedups, but the perception gap finding remains the most cited result.
metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- 3 Stack Overflow.** 2025 Developer Survey. 49,000 plus respondents across 177 countries, fielded May to June 2025. Headline figures cited in this guide: 84 per cent AI tool adoption, 29 per cent trust in AI accuracy (down from 40 per cent in 2024), 66 per cent frustrated by “almost right but not quite” output, 75 per cent would still ask a person for help when they do not trust AI's answers.
survey.stackoverflow.co/2025
- 4 Veracode.** 2025 GenAI Code Security Report. Tested over 100 large language models on 80 coding tasks across Java, JavaScript, Python and C#. Headline finding: 45 per cent of AI-generated code introduced vulnerabilities aligned with the OWASP Top 10. Java showed a 72 per cent security failure rate. Cross-site scripting failed in 86 per cent of cases.
veracode.com/resources/analyst-reports/2025-genai-code-security-report
- 5 HackerRank.** 2025 Developer Skills Report. Survey of 13,732 developers, managers and recruiters across 102 countries, combined with platform data from 26 million developers and 3 million annual assessments. Headline figures cited: 74 per cent of developers say finding a job is difficult, with junior hiring lagging behind more senior roles, partly due to employer concerns about coding ability without AI assistance.
hackerrank.com/reports/developer-skills-report-2025

All sources consulted in May 2026. Where figures vary slightly across sources or have been updated, the original published figure has been used in the body of the guide. Numbers in the AI productivity space are evolving fast: when in doubt, go to the original source and read the methodology.

Prepare

Foundations to lock in and projects to build before you apply. Stand-alone reference.

FOUNDATIONS TO LOCK IN

I. PROGRAMMING LANGUAGE - Python or JavaScript/TypeScript - Read code without copy-pasting	II. VERSION CONTROL (GIT) - Branch, commit, merge fluently - Resolve a merge conflict alone
III. TERMINAL & CLI - Navigate, find, search, edit - Be comfortable, not panicked	IV. HTTP & APIS - Request and response cycle - Call an external API from code
V. SQL & DATA - SELECT, JOIN, GROUP BY without lookup - Query the right answer, not just any	VI. TESTING - Unit, integration, why each exists - Write a test that catches a real bug

BUILD BEFORE YOU APPLY

PROJECT 1 SOLVE A REAL PROBLEM - Pick something annoying in your life - Build the smallest tool that fixes it - Ship it, even if only you use it	PROJECT 2 CONTRIBUTE TO OPEN SOURCE - Find a project you actually use - Open a small PR (doc, test, fix) - Get one merged, then another	PROJECT 3 REBUILD SOMETHING THAT EXISTS - Pick a small app you admire - Recreate the core feature from scratch - Compare your version to the original
---	--	--

NON-NEGOTIABLE HABITS

Verify everything. Confident output is not correct output.

Read more code than you write. Reviewing builds judgement faster than producing.

Stay curious about the business. Domain knowledge compounds. Languages do not.

Ship the thing. A finished small project beats an unfinished ambitious one.

Choose

What the data says, what good companies look like, and questions to ask in interviews.

BY THE NUMBERS

WHAT THE RESEARCH SHOWS

+15%

software developer job growth, 2024 to 2034

US BLS, 2025

84% / 29%

use AI tools / trust their accuracy

Stack Overflow, 2025

19%

slower: experienced devs using AI on their own code

METR, 2025

45%

of AI-generated code introduces an OWASP vulnerability

Veracode, 2025

READ THE ROOM: WHAT GOOD COMPANIES LOOK LIKE

LOOK FOR THIS

- + Code review treated as primary work, not a chore
- + Talks openly about how it uses agents
- + Real onboarding plan for graduates
- + Senior engineers spend time with juniors
- + “Verify, then ship” not “ship, then verify”

WALK AWAY FROM

- “Just ship it, agents fix it later”
- No clear policy on agent-generated code
- Speed metrics with no quality counterweight
- Code reviews that are rubber stamps
- Junior salaries with senior expectations

FIVE QUESTIONS TO ASK IN YOUR INTERVIEW

USE THESE TO DIAGNOSE THE TEAM YOU ARE JOINING

1. **How do you use AI tools in your team, and how do you verify the output?** Listen for specifics. Vague answers signal a team that has not thought it through.
2. **What does code review look like here?** The honest answer tells you everything about the engineering culture.
3. **What does the first year look like for a graduate on your team?** If they have no answer, they have no plan.
4. **Who would be my mentor, and how often would we meet?** Mentorship is a leading indicator of how you will grow.
5. **How do you decide what to ship and what to verify further?** Where the rubber meets the road on quality.